1 Лабораторная работа № 2 по курсу дискретного анализа: Сбалансированные деревья

Выполнил студент группы М8О-308б-22 МАИ *Караев Тариел Жоомартбекович.*

1.1 Условие

Реализовать декартово дерево с возможностью поиска, добавления и удаления элементов.

Необходимо создать программную библиотеку, реализующую указанную структуру данных, на основе которой разработать программу-словарь. В словаре каждому ключу, представляющему из себя регистронезависимую последовательность букв английского алфавита длиной не более 256 символов, поставлен в соответствие некоторый номер, от 0 до 264 - 1. Разным словам может быть поставлен в соответствие один и тот же номер.

Программа должна обрабатывать строки входного файла до его окончания. Каждая строка может иметь следующий формат:

+ word 34 — добавить слово «word» с номером 34 в словарь. Программа должна вывести строку «OK», если операция прошла успешно, «Exist», если слово уже находится в словаре.

- word — удалить слово «word» из словаря. Программа должна вывести «OK», если слово существовало и было удалено, «NoSuchWord», если слово в словаре не было найдено.

word — найти в словаре слово «word». Программа должна вывести «OK: 34», если слово было найдено; число, которое следует за «OK:» — номер, присвоенный слову при добавлении. В случае, если слово в словаре не было обнаружено, нужно вывести строку «NoSuchWord»

1.2 Метод решения

Для решения задачи был использован алгоритм декартового дерева (treap), которое сочетает в себе свойства бинарного дерева поиска и структуры данных на куче. Каждый узел дерева содержит ключ и приоритет, где ключи организованы по правилам бинарного дерева поиска, а приоритеты — по свойствам кучи. Это позволяет эффективно выполнять операции вставки, удаления и поиска элементов. В рамках задачи был разработан словарь, в котором ключами являются слова, а значениями — присвоенные им номера.

1.3 Описание программы

main.cpp — содержит основную логику работы программы, включая функции для добавления, удаления и поиска слов в словаре, а также чтение и запись в файлы.

node — структура данных, представляющая узел декартового дерева. Каждый узел хранит:

- key — ключ (слово), строка длиной до 256 символов;
- value — значение, соответствующее ключу;
- priority — приоритет для обеспечения свойств кучи;
- left и right — указатели на левые и правые дочерние узлы.
- split(node root, node left, node right, const char key) — рекурсивно разделяет дерево по ключу key на два поддерева: одно содержит все ключи меньше либо равные данному, другое — больше.
- merge(node left, node right) — объединяет два поддерева в одно, сохраняя свойства декартового дерева.
- insert(node root, node item) — вставляет новый узел в дерево, соблюдая свойства кучи и бинарного дерева поиска.
- remove(node root, const char key) — удаляет узел с данным ключом из дерева.
- search(node root, const char key) — выполняет поиск узла по ключу в дереве.
- toLower(char str) — переводит строку в нижний регистр для обеспечения регистронезависимого поиска.
- destroy(node node) — рекурсивно освобождает память, занятую деревом.

1.4 Дневник отладки

В основном возникали проблемы с компиляцией в виду неверного выбора комплиятора.

1.5 Тест производительности

Код бенчмарка:

```
#include <iostream>
#include <map>
#include <chrono>
#include <random>
#include <string>
#include <algorithm>
#include <cstdint>
#include <cstring>
#include <fstream>
#include <cstdlib>
```

```
#include <cctype>
#include "main.cpp"

int main() {
    // Генератор случайных чисел и строк
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_int_distribution<> dis(0, 25);
    std::uniform_int_distribution<> num(0, 1000000);

    const int testSize = 1000;
    std::vector<std::string> words;
    for (int i = 0; i < testSize; ++i) {
        std::string word;
        for (int j = 0; j < 10; ++j) {
            word += 'a' + dis(gen);
        }
        words.push_back(word);
    }

    // Бенчмарк декартового дерева
    node *root = nullptr;
    auto start = std::chrono::high_resolution_clock::now();

    for (const auto &word : words) {
        insert(root, new node(word.c_str(), num(gen)));
    }
    auto end = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> insertTimeTreap = end - start;

    start = std::chrono::high_resolution_clock::now();
    for (const auto &word : words) {
        search(root, word.c_str());
    }
    end = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> searchTimeTreap = end - start;

    start = std::chrono::high_resolution_clock::now();
    for (const auto &word : words) {
        remove(root, word.c_str());
    }
    end = std::chrono::high_resolution_clock::now();
```

```
    std::chrono::duration<double> removeTimeTreap = end - start;

    // Бенчмарк std::map
    std::map<std::string, uint64_t> map;
    start = std::chrono::high_resolution_clock::now();
    for (const auto &word : words) {
        map[word] = num(gen);
    }
    end = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> insertTimeMap = end - start;

    start = std::chrono::high_resolution_clock::now();
    for (const auto &word : words) {
        map.find(word);
    }
    end = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> searchTimeMap = end - start;

    start = std::chrono::high_resolution_clock::now();
    for (const auto &word : words) {
        map.erase(word);
    }
    end = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> removeTimeMap = end - start;

    std::cout << "Treap Insertion: " << insertTimeTreap.count() << " seconds\n";
    std::cout << "Treap Search: " << searchTimeTreap.count() << " seconds\n";
    std::cout << "Treap Removal: " << removeTimeTreap.count() << " seconds\n";

    std::cout << "Map Insertion: " << insertTimeMap.count() << " seconds\n";
    std::cout << "Map Search: " << searchTimeMap.count() << " seconds\n";
    std::cout << "Map Removal: " << removeTimeMap.count() << " seconds\n";

    destroy(root);

    return 0;
}
```

Результат скорости работы при 1000000 входных данных:

```
Treap Insertion: 1.99359 seconds
Treap Search: 1.69025 seconds
Treap Removal: 1.47075 seconds
Map Insertion: 1.43052 seconds
```

```
Map Search: 1.25367 seconds
Map Removal: 1.5627 seconds
```

Мы видим, что скорость удаления у декартового дерева выше, но в остальном выигрывает std::map. Это обуславливается более стабильной работой красно-чёрного дерева по сравнению с декартовым деревом, в котором присутствует определённая случайность приоритетов.

1.6 Недочёты

При вставке и поиске декартово дерево показывает результат немного медленее std::map.

1.7 Выводы

В ходе лабораторной работы было реализовано декартово дерево, которое эффективно решает задачи вставки, удаления и поиска с логарифмической сложностью. Данная структура применяется для управления упорядоченными данными и поддержания динамических множеств. Основные сложности возникли при реализации процедур слияния и разделения дерева для поддержания случайной балансировки. По сравнению с std::map, декартово дерево показало конкурентоспособную производительность.